CS637 : Embedded and Cyber Physical System

Path-Oriented, Derivative-Free Approach for Safety Falsification of Nonlinear and Nondeterministic CPS

Instructor : Prof: Indranil Saha

Team Members

- ❖ Himanshu Karnatak - 231110017
- ❖ LtCdr Sunil Kumar - 231110025
- ❖ Priyanshu Chaurasiya -210780
- ❖ Pradeep Kumar Bagri -210734
- ❖ Ashish Saini - 210212
- ❖ Harsh Khandelwal - 210407
- ❖ Palak Parashar -210691

Agenda

- ➢ Introduction
- ➢ Problem Formulation
- ➢ Classification Model-Based Derivative-Free Optimization
- ➢ Path Oriented DFO Based Falsification
- ➢ Classification model based DFO Algorithm
- ➢ Nested Algorithm
- ➢ Shortest Hardly feasible Prefix Generation
- ➢ Implementation And Evaluation
- ➢ Related Work
- ➢ Conclusion
- ➢ Implementation
- ➢ Critical Analysis
- ➢ Contribution

Introduction

- Verifications of CPS.
- Alternative approach.
- Falsification of HA.
  - Motion-planning-based falsification
  - Optimization-based falsification
- Dynamic of many complex CPS in the real world.
- To handle this large search space, applied a two-layered path-oriented framework.
- Two lightweight techniques to prune the search space on the continuous and discrete levels.

Definitions

Hybrid Automata:

H=(L,X,U,E,F,Inv,G,R,S0,Sf)

Semantics: Continuous Evolution ,Discrete Evolution

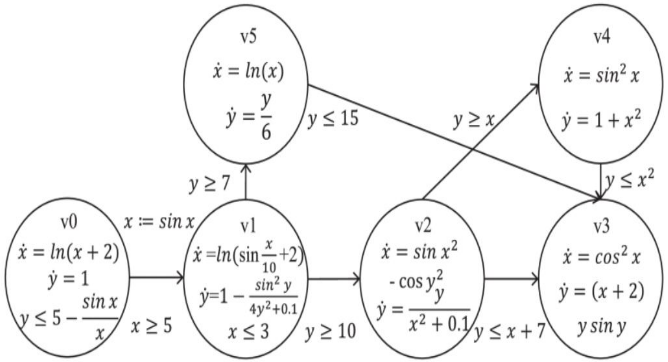
Path:

$$\rho = \{l_i\}_0^N$$

Jumping Time Sequence:

$$\tau = \{t_i\}_0^{N+1}$$

Control Instance: δ=(ρ,τ,μ,x)



Definitions

Simulated Trajectory: Collection of Differentiable maps

1) *Flow:* $\forall t \in [t_i, t_{i+1}),\ \mathcal{X}_\delta^i(t) = F_{l_i}(\mathcal{X}_\delta^i(t), \mu(t))$.

2) *Reset:* $\mathcal{X}_\delta^{i+1}(t_{i+1}) = R_{(l_i, l_{i+1})}(\mathcal{X}_\delta^i(t_{i+1}), \mu(t_{i+1}))$.

Feasible Trajectory: ρ is feasible if all locations in ρ are reachable

Witness Trajectory:

1) $(l_0, \mathcal{X}_\delta^0(t_0)) \in S_0$;

2) $(l_N, \mathcal{X}_\delta^N(t_{N+1})) \in S_f$.

Classification Model-Based Derivative-Free Optimization

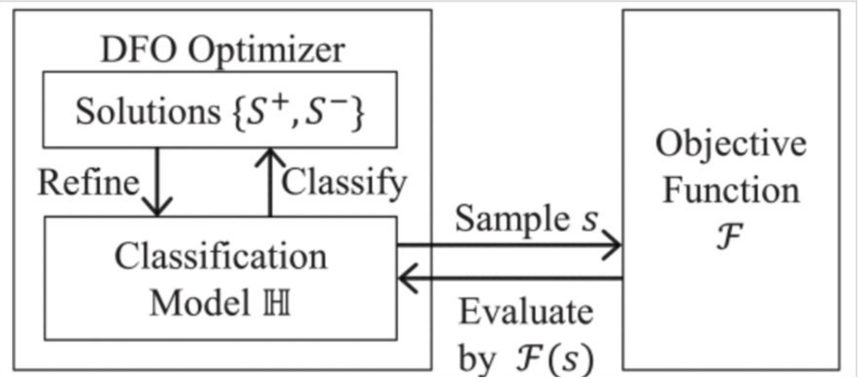
The Objective function F gives min F(x) for x ϵ X (domain).

Many DFO are heuristic based , it uses distribution bases probability algorithm to find the optimum solution.

This method uses statistical technique like Bayesian optimisation or evolutionary algo, where traditional method becomes very cumbersome and expensive.

Sampling & model refinement cycles iterates to improve the accuracy of classification model H. It learns over each iteration like a model and discriminate bad solutions from good ones.

In each iteration classification based DFO samples a batch of new solutions S from the domain H.



Path Oriented DFO Based Falsification

Behavior of a nondeterministic HA is decided by a control instance $\delta = (\rho, \tau, \mu, x)$

Where ρ : path, τ : jumping time sequence , μ : External inputs X : initial continuous state.

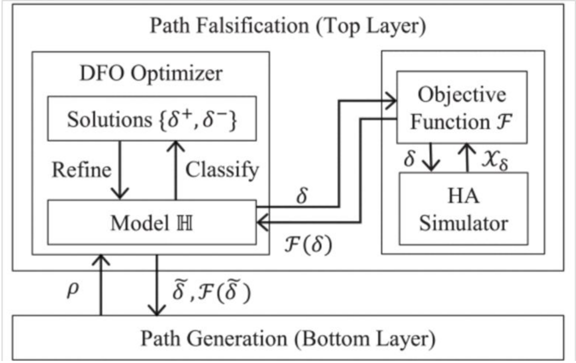
P generates from So (initial state) $\longrightarrow$ S_f(final state).

Generate a graph structure of HA by its discrete locations & transition use DFS to reach target & generate all possible paths.

Encode falsification related feasibility for each path and define F for this path.

For a given simulated trajectory $X\delta$ for path ρ we encode all constraints on ρ so that $X\delta$ should satisfy if it is a witness trajectory.

X_δ should satisfy conditions in S_0 and S_f , guard condition of each discrete transition along the path ρ should be satisfied .



$$\mathbb{C}_{\text{dis}}(\delta) = S_0\left(l_0, \mathcal{X}_\delta^0(t_0)\right) \bigwedge S_f\left(l_N, \mathcal{X}_\delta^N(t_{N+1})\right) \bigwedge_{i\in[0,N), i\in\mathbb{Z}} G_{(l_i,l_{i+1})}\left(\mathcal{X}_\delta^i(t_{i+1}), \mu(t_{i+1})\right).$$

$$\mathbb{C}_{\text{con}}(\delta) = \bigwedge_{i\in[0,N], i\in\mathbb{Z}} \bigwedge_{\forall t\in[t_i,t_{i+1})} \text{Inv}_{l_i}\left(\mathcal{X}_\delta^i(t), \mu(t)\right).$$

$$\mathbb{C}(\delta) = \mathbb{C}_{\text{dis}}(\delta) \bigwedge \mathbb{C}_{\text{con}}(\delta).$$

Once we find a potential solution δ of a constraint $C(\delta)$ we will find its dissatisfactory degree

$$F(\delta) = D(C(\delta))$$

A constraint can be of the from Expr $\bowtie 0$ where $\bowtie \in \{\geq, >, =, \neq\}$.

Function “**val**”: Expr $\rightarrow$R

The dis-satisfactory degree is given as below

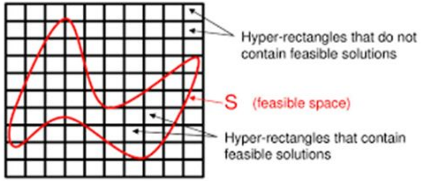
$$\mathcal{D}(\mathbb{C}) = \begin{cases} \mathbf{val}(\text{Expr}) \geq 0 ?\ 0 : -\mathbf{value}(\text{Expr}), & \text{if } \mathbb{C} \text{ is Expr } \geq 0 \\ \mathbf{val}(\text{Expr}) > 0 ?\ 0 : 1 - \mathbf{value}(\text{Expr}), & \text{if } \mathbb{C} \text{ is Expr} > 0 \\ \mathbf{val}(\text{Expr}) = 0 ?\ 0 : |\mathbf{value}(\text{Expr})|, & \text{if } \mathbb{C} \text{ is Expr} = 0 \\ \mathbf{val}(\text{Expr}) \neq 0 ?\ 0 : 1, & \text{if } \mathbb{C} \text{ is Expr} \neq 0. \end{cases}$$

In case of complex which are conjunction or disjunction of simple or complex constraints the dissatisfactory degree will be given as below

$$\mathcal{D}(\mathbb{C}) = \begin{cases} \min(\mathcal{D}(\mathbb{C}_1), \mathcal{D}(\mathbb{C}_2)), & \text{if } \mathbb{C} = \mathbb{C}_1 \bigvee \mathbb{C}_2 \\ \mathcal{D}(\mathbb{C}_1) + \mathcal{D}(\mathbb{C}_2), & \text{if } \mathbb{C} = \mathbb{C}_1 \bigwedge \mathbb{C}_2. \end{cases}$$

Classification model based DFO

- Optimization
  - A classification model H
  - Each solution is good if it lies in H, else bad.
  - Iterate through mutations of a randomly selected control instance
  - Get optimal control instance for given path.
- Model Refinement
  - Initialize model by control instance domain
  - Repeat until S- is empty
  - For a random dimension d
  - Partition set S-
  - Shrink lower and upper bounds for dimensions in H
- Sampling
  - Generate new solution from H by mutating control instance ‘n’ times.
  - A random dimension every time. Value = random within limits in H



$\rho_0 = v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow v_3$	3.814
$\rho_1 = v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow v_4 \rightarrow v_3$	5.591
$\rho_2 = v_0 \rightarrow v_1 \rightarrow v_5 \rightarrow v_3$	0

Algorithm 1 Classification Model-Based DFO for the Optimal Control Instance Along a Candidate Path

Input:
X : Domain of Control Instances along the path ρ;
$\mathcal{F}$: Dissatisfactory Degree Function of the path ρ;
N : Number of Iterations;
m : Number of Solutions Sampled in Each Iteration;
n : Number of Mutations When Sampling New Solutions.

Output: $\arg\min_{\delta \in X} \mathcal{F}(\delta)$, $\min_{\delta \in X} \mathcal{F}(\delta)$.

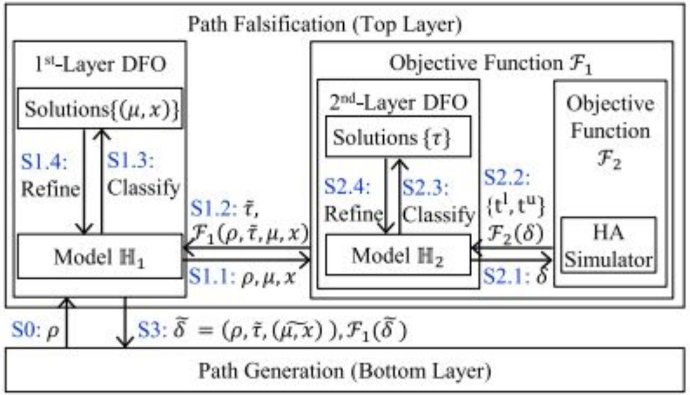
```
function OPTIMIZE(X, F, N, m, n)
    S, Y = {}                                  ▷ Solution set, evaluation set
    S⁺ = X, S⁻ = {}                            ▷ Positive, negative solutions
    δ̄ = {}                                     ▷ Optimal solution
    for n = 1 to N do
        for i = 1 to m do
            δ⁺ = UniformRandom(S⁺)
            ℍ = MODELREFINE(X, δ⁺, S⁻)
            δ_i = SAMPLE(ℍ, δ⁺, n, |X|)
            y_i = F(δ_i)                       ▷ Evaluate solutions
        end for
        S = {δ_1, ..., δ_m}, Y = {y_1, ..., y_m}
        {S⁺, S⁻} = C_k(S, Y)                   ▷ Classify solutions
        δ̄ = arg min_{δ∈S∪{δ̄}} F(δ)             ▷ Update optimal solution
    end for
    return δ̄, F(δ̄)                             ▷ Optimal solution and evluation
end function

function MODELREFINE(X, δ⁺, S⁻)
    ℍ = X                                      ▷ Init new model
    while ∃δ⁻ ∈ S⁻, s.t. δ⁻ ∈ ℍ do
        d = UniformRandom(1, |X|)              ▷ Select dimension
        S_l⁻ = {δ⁻ ∈ S⁻ | δ⁻[d] > δ⁺[d]}
        S_s⁻ = {δ⁻ ∈ S⁻ | δ⁻[d] < δ⁺[d]}
        if |S_l⁻| ≥ |S_s⁻| then                ▷ Shrink upper bound
            r = UniformRandom(δ⁺[d], min_{δ∈S_l⁻} s[d])
            ℍ = ℍ ∩ ℍ[d] ≤ r
        end if
        if |S_l⁻| ≤ |S_s⁻| then                ▷ Shrink lower bound
            r = UniformRandom(max_{δ∈S_s⁻} s[d], δ⁺[d])
            ℍ = ℍ ∩ ℍ[d] ≥ r
        end if
    end while
    return ℍ                                   ▷ Return learned model
end function

function SAMPLE(ℍ, δ⁺, n, D)
    δ = δ⁺                                     ▷ Init new solution
    for i = 1 to n do                          ▷ Mutate n times
        d = UniformRandom(1, D)                ▷ Select dimension
        δ[d] = UniformRandom(ℍ[d].low, ℍ[d].up)
    end for
    return δ                                   ▷ Return new solution
end function
```

Search Space Pruning: Technique 1

- Using jumping time bounds.
  - Upper bound: First moment when invariant is not satisfied.
  - Lower bound: First moment when guard is satisfied and invariant is not violated.
  - If lower bound doesn’t exist, location doesn’t exist in this control instance
- Better model refinement
  - Reducing jumping time sequences
  - Two layer approach: Nested DFO



Layer1
- Samples and iterates external inputs and initial values.
- Gets optimal jumping time sequence and dissatisfactory degree form layer2.
- Model H1 is refined for inputs and initial values

Layer2
- Samples and evaluates time sequences iteratively
- Jumping time bounds for each control instance obtained by HA simulator
- Jumping time bounds refine model H2 more efficiently

Search Space Pruning: Technique 2

- **Problem:** In previous methods,the problem being addressed is the complexity of searching through the discrete search space of a Hybrid Automaton, which can be very large and systems with a large search space, leads to issues known as "path explosion."
- **Objective:**
  - Improve the efficiency of finding counterexamples or witnesses in Hybrid Automaton of Discrete search space.
  - Identify problematic areas where conditions cannot be met as required by the properties being verified.
- **Key Concepts:**
  - **Infeasible Location Sets:** These are sets of locations within a system where it is impossible to satisfy the constraints all at once. When searching for a counterexample related to a specific property, these sets can be instrumental in pinpointing problematic areas.
  - **Infeasible Path Prefix:** The segment of a path containing an infeasible location set, making constraint satisfaction impossible.
  - **Solution: The Shortest Hardly Feasible Path Prefix**
  - The "Shortest Hardly Feasible Path Prefix" is designed as an approximation to the more complex "Shortest Infeasible Path Prefix" with the aim of simplifying the process of pruning the discrete search space.
  - This abbreviated infeasible path prefix serves as a valuable aid in guiding the backtracking process during path generation, facilitating more efficient exploration of the system's possibilities.
  - **Benefits:**
  - Efficiently guides backtracking during path generation.
  - Avoids generating and checking unnecessary candidate paths.
  - Helps in identifying problematic regions within the system model.

Algorithm 2: Shortest Hardly Feasible Prefix Generation

- They perform backtracking guided by this path prefix, during our path generation process on the bottom layer.

- If the simulation limit is reached and no witness is found in the current candidate path, the algorithm takes an alternative approach.

- Instead of generating and checking the next candidate path in the Depth-First Search (DFS) order, the algorithm directly focuses on examining a more suitable option.

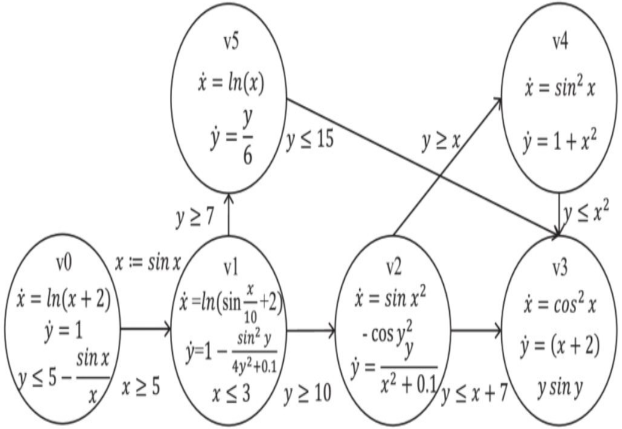
Algorithm 2 Shortest Hardly Feasible Prefix Generation

Input:
ρ : the Candidate Path;
N: the Number of Simulations.
Output:
$\{l_i\}_0^{\text{index}}$: the Shortest Hardly Feasible Path Prefix of ρ.

```
 1: function GENPREFIX(ρ, N)
 2:     index = |ρ| − 1                    ▷ Init it to the index of last location
 3:     for i = 1 to 3 do                  ▷ Check location sets with size 1-3
 4:         for each set λ, |λ| = i, λ ⊆ ρ do
 5:             sat=FALSE                  ▷ Init satisfiability
 6:             for j = 0 to N-1 do        ▷ Check all simulations
 7:                 if all the constraints w.r.t. λ is TRUE then
 8:                     sat=TRUE, goto line 11
 9:                 end if
10:             end for
11:             if ¬ sat then              ▷ Update to the smallest one
12:                 index = min(index, λ[i − 1])
13:             end if
14:         end for
15:     end for
16:     return {l_i}_0^index               ▷ The shortest hardly feasible prefix
17: end function
```

Example

- Candidate Path, **ρ0 : v0 → v1 → v2 → v3**
- Hardly Feasible Location Sets:
  λ0 = {v0, v1, v2} and λ1 = {v2, v3}
- Shortest Hardly Feasible Path Prefix:
  ρ'0 = v0 → v1 → v2 (v2 has the smallest index)
- It moves from the last location of the original path, v3, back to a previous location, v1, on the lower layer of the HA.
- This backtracking leads to the generation of a new candidate path,
  ρ2 = v0 → v1 → v5 → v3
- By running algorithm, ρ2 is falsified because it meets all the desired conditions and properties.
- And hence the shortest hardly feasible path prefix is v0 → v1 → v2



Implementation

- This approach was implemented as a tool called PDF (path-oriented, derivative-free falsification) in C++.
- Given an Hybrid Automata, PDF would return a witness trajectory if it finds any within the given simulation iterations.

System parameters of PDF:

- Number of control points for external inputs
- The determinism of Hybrid Automata
- The error tolerance
- The simulation time horizon
- The simulation iteration bound for each path
- The total simulation iteration bound

Benchmarks

- Nonlinearity and nondeterminism
- Different numbers of discrete locations
- Continuous states and external inputs
- Different difficulty levels of unsafe specifications to be falsified.

For example, The AFC1 and AFC2 benchmark , consists of a powertrain engine model and an air-fuel controller with four operation modes, two external inputs, and eight continuous states.

In this experiment, falsification of two requirements that can be written in the form of safety property, with the input settings was done.

To be specific, we fixed the input throttle to be piecewise constant with ten uniform segments over [0, 61.2) and the input speed to be constant over [900, 1100).

AFC1a-c falsifies the controlled signal overshoot/ undershoot requirement and AFC2a-c falsifies the accumulated error requirement presented in there.

TABLE I

EXPERIMENTAL RESULTS OF PDF WITH AND WITHOUT PRUNING TECHNIQUES IN ALL BENCHMARKS

Benchmark							Success Rate(%)		#Iter[2]		Time(s)[2]	
Name	Description	Det[1]	#Loc[1]	#Var[1]	#Mu[1]	Unsafe Conditions	Basic[3]	Opt[3]	Basic	Opt	Basic	Opt
Air-1	Aircraft [35]	True	1	3	2*10	v1, $x_1 \in [260, \infty), x_2 \in [-10, 10], t \in [3, 3.2]$	**100**	**100**	502	**399**	**0.2**	0.5
Air-2						v1, $(x_1 + x_3) \in [413.5, \infty), t \in [3, 4]$	13	**64**	**2075**	3643	**1.3**	3.6
Air-3						v1, $x_1 \in [256, \infty), x_3 \in [146, \infty), t \in [3, 4]$	20	**32**	6675	**6560**	**4.3**	10.2
IG-1	Insulin Glucose Control (IG) [36]	True	6	4	0	v3, $G \in [-\infty, 5.325], t \in [88, 92]$	0	**100**	-	**490**	-	**0.7**
IG-2						v5, $G \in [-\infty, 4], I \in [-\infty, -13.29], t \in [120, 360]$	0	**100**	-	**565**	-	**2.6**
IG-3						v3, $G \in [-\infty, 5.52], t \in [98, 102]$	1	**100**	4594	**2226**	4.9	**3.4**
IG-4	Modified IG [36]	True	6	4	1*5	v4, $G \in [6, 16], t \in [90, 100]$	0	**100**	-	**296**	-	**0.4**
Nav-1	Navigation [15]	True	16	4	0	v15,$x \in [2.47, 2.53], v_x \in [-\infty, 0.8], v_y \in [-0.5, 0.5], t \in [0, 10]$	0	**100**	-	**1013**	-	**0.5**
Nav-2						v12, $x \in [3.1, 3.9], y \in [2.1, 2.9], t \in [0, 25]$	0	**97**	-	**4486**	-	**3.4**
Nav-3	Navigation [37]	True	25	4	0	v6,$(x + y) \in [2.399, 2.401], t \in [0, 5]$	1	**100**	2186	**150**	0.2	**0.1**
Nav-4						v6, $x \in [0.99, 1], y \in [1.3, 1.6], v_x \in [0, 1], \lvert v_y \rvert \in [0, 0.2], t \in [0, 5]$	0	**100**	-	**897**	-	**0.2**
Nav-5	Navigation [37]	True	225	4	0	v125, $x \in [4, 4.96], t \in [0, 5]$	0	**89**	-	**2864**	-	**2.1**
Nav-6	Modified Navigation [15]	False	16	4	2*5	v11, $x \in [2.5, 3], t \in [0, 25]$	**100**	**100**	1338	**790**	0.6	**0.3**
Nav-7						v12, $t \in [0, 25]$	73	**95**	3428	**1903**	2.0	**1.0**
Nav-8						v8, $y \in [1.6, 2], t \in [0, 25]$	38	**95**	**5383**	16308	**3.9**	17.4
Nav-9						v3, $x \in [2.3, 2.7], t \in [0, 25]$	76	**95**	24267	**20739**	**17.3**	19.2
Nav-10	Modified Navigation [37]	False	25	4	2*5	v3, $x \in [2.3, 2.7], t \in [0, 25]$	**100**	**100**	1078	**63**	0.4	**0.1**
Nav-11						v5, $v_x, v_y \in [-1, 1], t \in [0, 25]$	87	**96**	3315	**2957**	**1.7**	2.1
Nav-12						v10, $x \in [4.2, 5], t \in [0, 25]$	76	**80**	6927	**6629**	**4.3**	5.2
Nav-13	Modified Navigation [37]	False	225	4	2*5	v125, $y - x \in [3, 4], t \in [0, 25]$	90	**99**	9992	**6960**	5.8	**5.3**
Nav-14						v70, $t \in [0, 25]$	49	**99**	**11489**	15938	**7.2**	11.4
Nav-15						v71, $t \in [0, 25]$	21	**50**	**13804**	16524	**10.2**	15.5
AFC-1a	Modified Automotive Powertrain [42]	True	4	8	1*10 +1*1	v2, $\lvert\mu\rvert \in [0.0075, \infty], t \in [11, 50]$	82	**100**	4668	**2**	165.9	**0.1**
AFC-1b						v2, $\lvert\mu\rvert \in [0.0076, \infty], t \in [11, 50]$	81	**100**	4122	**20**	151.6	**1.7**
AFC-1c						v2, $\lvert\mu\rvert \in [0.0077, \infty], t \in [11, 50]$	77	**100**	4680	**31**	165.4	**2.7**
AFC-2a						v2, $\mathbf{x}_{rms} \in [0.30, \infty], t \in [11, 50]$	9	**100**	5221	**99**	314.7	**11.4**
AFC-2b						v2, $\mathbf{x}_{rms} \in [0.31, \infty], t \in [11, 50]$	8	**100**	6223	**146**	385.5	**17.0**
AFC-2c						v2, $\mathbf{x}_{rms} \in [0.32, \infty], t \in [11, 50]$	5	**96**	6331	**1310**	396.3	**159.7**

[1] Det marks the determinism of systems, #Loc is the number of discrete locations, #Var is the number of continuous states, #Mu is the number of external inputs times the number of control points.

[2] #Iter is the average simulation iterations for successfully falsified runs. Time(s) is the average execution time for successfully falsified runs in seconds.

[3] PDF was run with 2 settings: Basic (without pruning technique) and Opt (with both pruning techniques).

Simulation

- Used Sundials’ CVODE to handle the numerical integration of the ordinary differential equations associated with the flow functions in the HA.

- The interpolation of the external inputs is obtained by GNU scientific library (GSL), which offers various interpolation types and methods to the users.

Experimental Setup

Our experiments are organized in the following aspects:

- The performance of our approach was evaluated and the effectiveness of the two pruning techniques proposed under our path-oriented, DFO-based framework.
- The performance of PDF was evaluated by comparing it with S-TaLiRo and Breach, two state-of-the-art optimization-based falsification tools designed to falsify temporal logic specifications in deterministic Hybrid Automata
- Therefore, we compared PDF with them merely in deterministic benchmarks. We converted the unsafe conditions to safety properties expressed by temporal logic formulas to translate all models into Simulink models so that Breach can handle.
- For the AFC benchmark, we used the HA version of the AFC system presented in.
- We implemented the SA algorithm in PDF according to S-TaLiRo's setting. Then, we compared the performance of our classification model-based DFO algorithm with this classical heuristic method.

Evaluations :

- **Evaluation of PDF and Its Pruning Techniques**

- Successful generation of witness trajectories for all 28 benchmarks with success rates no lower than 95% in 23 cases.
- Efficient witness trajectory generation in seconds for all benchmarks, with only 2 cases taking more than 18 seconds.
- Pruning techniques significantly improved success rates, solving previously failed cases with success rates no lower than 89%.
- Improved performance seen in both success rates and efficiency when comparing the basic and optimized versions of PDF.

Benchmark							Success Rate(%)		#Iter[2]		Time(s)[2]	
Name	Description	Det[1]	#Loc[1]	#Var[1]	#Mu[1]	Unsafe Conditions	Basic[3]	Opt[3]	Basic	Opt	Basic	Opt
Air-1	Aircraft [35]	True	1	3	2*10	v1, $x_1 \in [260, \infty), x_2 \in [-10, 10], t \in [3, 3.2]$	**100**	**100**	502	**399**	**0.2**	0.5
Air-2						v1, $(x_1 + x_3) \in [413.5, \infty), t \in [3, 4]$	13	**64**	**2075**	3643	**1.3**	3.6
Air-3						v1, $x_1 \in [256, \infty), x_3 \in [146, \infty), t \in [3, 4]$	20	**32**	6675	**6560**	**4.3**	10.2
IG-1	Insulin Glucose Control (IG) [36]	True	6	4	0	v3, $G \in [-\infty, 5.325], t \in [88, 92]$	0	**100**	-	**490**	-	**0.7**
IG-2						v5, $G \in [-\infty, 4], I \in [-\infty, -13.29], t \in [120, 360]$	0	**100**	-	**565**	-	**2.6**
IG-3						v3, $G \in [-\infty, 5.52], t \in [98, 102]$	1	**100**	4594	**2226**	4.9	**3.4**
IG-4	Modified IG [36]	True	6	4	1*5	v4, $G \in [6, 16], t \in [90, 100]$	0	**100**	-	**296**	-	**0.4**
Nav-1	Navigation [15]	True	16	4	0	v15,$x \in [2.47, 2.53], v_x \in [-\infty, 0.8], v_y \in [-0.5, 0.5], t \in [0, 10]$	0	**100**	-	**1013**	-	**0.5**
Nav-2						v12, $x \in [3.1, 3.9], y \in [2.1, 2.9], t \in [0, 25]$	0	**97**	-	**4486**	-	**3.4**
Nav-3	Navigation [37]	True	25	4	0	v6,$(x + y) \in [2.399, 2.401], t \in [0, 5]$	1	**100**	2186	**150**	0.2	**0.1**
Nav-4						v6, $x \in [0.99, 1], y \in [1.3, 1.6], v_x \in [0, 1], \lvert v_y \rvert \in [0, 0.2], t \in [0, 5]$	0	**100**	-	**897**	-	**0.2**
Nav-5	Navigation [37]	True	225	4	0	v125, $x \in [4, 4.96], t \in [0, 5]$	0	**89**	-	**2864**	-	**2.1**
Nav-6	Modified Navigation [15]	False	16	4	2*5	v11, $x \in [2.5, 3], t \in [0, 25]$	**100**	**100**	1338	**790**	0.6	**0.3**
Nav-7						v12, $t \in [0, 25]$	73	**95**	3428	**1903**	2.0	**1.0**
Nav-8						v8, $y \in [1.6, 2], t \in [0, 25]$	38	**95**	**5383**	16308	**3.9**	17.4
Nav-9						v3, $x \in [2.3, 2.7], t \in [0, 25]$	76	**95**	24267	**20739**	**17.3**	19.2
Nav-10	Modified Navigation [37]	False	25	4	2*5	v3, $x \in [2.3, 2.7], t \in [0, 25]$	**100**	**100**	1078	**63**	0.4	**0.1**
Nav-11						v5, $v_x, v_y \in [-1, 1], t \in [0, 25]$	87	**96**	3315	**2957**	**1.7**	2.1
Nav-12						v10, $x \in [4.2, 5], t \in [0, 25]$	76	**80**	6927	**6629**	**4.3**	5.2
Nav-13	Modified Navigation [37]	False	225	4	2*5	v125, $y - x \in [3, 4], t \in [0, 25]$	90	**99**	9992	**6960**	5.8	**5.3**
Nav-14						v70, $t \in [0, 25]$	49	**99**	**11489**	15938	**7.2**	11.4
Nav-15						v71, $t \in [0, 25]$	21	**50**	**13804**	16524	**10.2**	15.5
AFC-1a	Modified Automotive Powertrain [42]	True	4	8	1*10 +1*1	v2, $\lvert\mu\rvert \in [0.0075, \infty), t \in [11, 50]$	82	**100**	4668	**2**	165.9	**0.1**
AFC-1b						v2, $\lvert\mu\rvert \in [0.0076, \infty), t \in [11, 50]$	81	**100**	4122	**20**	151.6	**1.7**
AFC-1c						v2, $\lvert\mu\rvert \in [0.0077, \infty), t \in [11, 50]$	77	**100**	4680	**31**	165.4	**2.7**
AFC-2a						v2, $\mathbf{x}_{rms} \in [0.30, \infty), t \in [11, 50]$	9	**100**	5221	**99**	314.7	**11.4**
AFC-2b						v2, $\mathbf{x}_{rms} \in [0.31, \infty), t \in [11, 50]$	8	**100**	6223	**146**	385.5	**17.0**
AFC-2c						v2, $\mathbf{x}_{rms} \in [0.32, \infty), t \in [11, 50]$	5	**96**	6331	**1310**	396.3	**159.7**

[1] Det marks the determinism of systems, #Loc is the number of discrete locations, #Var is the number of continuous states, #Mu is the number of external inputs times the number of control points.
[2] #Iter is the average simulation iterations for successfully falsified runs. Time(s) is the average execution time for successfully falsified runs in seconds.
[3] PDF was run with 2 settings: Basic (without pruning technique) and Opt (with both pruning techniques).

Experimental Results of PDF With and Without Pruning Techniques in All Benchmarks

Cntd.

- **Comparison of PDF With Other Tools in Deterministic Cases:**

- PDF outperformed S-TaLiRo and Breach in success rates, solving all 18 deterministic cases with high success rates, especially in 17 cases.
- PDF required the best iteration numbers in 17 of the 18 cases compared to S-TaLiRo and Breach.
- PDF generally consumed less time compared to S-TaLiRo and Breach, showing better performance in most cases.
- Notable stability issues observed in the performance of Breach, especially when the corner sample strategy was ineffective.

Benchmark Name	Success Rate(%)					#Iter[1]					Time(s)[1]				
	PDF	S-TaLiRo			Breach	PDF	S-TaLiRo			Breach	PDF	S-TaLiRo			Breach
		SA	SOAR	CE	NM		SA	SOAR	CE	NM		SA	SOAR	CE	NM
Air-1	**100**	0	**100**	**100**	30	399	-	**236**	512	1528	**0.5**	-	127.5	8.5	757.6
Air-2	**64**	0	30	5	0	3643	-	**286**	2464	-	**3.6**	-	1346.3	51.8	-
Air-3	**32**	0	30	0	0	6560	-	**271**	-	-	**10.2**	-	961.9	-	-
IG-1	**100**	99	**100**	90	**100**	490	962	103	3626	**2**	0.7	4.0	304.0	13.9	**0.3**
IG-2	**100**	**100**	**100**	**100**	0	565	338	**52**	1004	-	**2.6**	53.7	358.8	414.6	-
IG-3	**100**	**100**	**100**	60	**100**	2226	2176	114	4207	**2**	3.4	8.0	331.1	14.2	**0.4**
IG-4	**100**	**100**	**100**	**100**	**100**	296	547	81	647	**3**	**0.4**	8.9	12.3	10.1	1.7
Nav-1	**100**	**100**	15	43	**100**	1013	1091	**207**	599	1870	**0.5**	93.8	399.7	44.1	466.1
Nav-2	97	29	10	0	**100**	4486	4938	1369	-	**2**	3.4	960.2	1096.2	-	**0.6**
Nav-3	**100**	**100**	90	38	**100**	150	1118	232	2403	**13**	**0.1**	111.2	232.3	168.6	4.5
Nav-4	**100**	**100**	85	**100**	85	897	1041	**453**	2210	2284	**0.2**	99.5	500.7	148.3	765.0
Nav-5	**89**	86	0	17	50	2864	2398	-	6927	**327**	**2.1**	590.7	-	1547.6	938.4
AFC-1a	**100**	20	95	55	**100**	**2**	250	159	247	3	**0.1**	1118.2	745.3	995.7	3.6
AFC-1b	**100**	10	75	20	**100**	**20**	208	146	177	401	**1.7**	931.4	706.1	681.9	581.8
AFC-1c	**100**	0	55	0	70	**31**	-	156	-	460	**2.7**	-	783.3	-	1037.4
AFC-2a	**100**	0	80	0	**100**	99	-	157	-	**78**	**11.4**	-	843.2	-	115.9
AFC-2b	**100**	0	60	0	**100**	146	-	139	-	**78**	**17.0**	-	746.6	-	117.9
AFC-2c	**96**	0	40	0	0	1310	-	**152**	-	-	**159.7**	-	948.3	-	-

#Iter is the average simulation iterations for successfully falsified runs. Time(s) is the average execution time for successfully falsified runs in seconds.

Related Works

- Verification techniques have advanced but struggle with complexity, scalability, and handling nonlinear, nondeterministic hybrid systems.
- Falsification has gained importance as a practical solution to address these limitations.

Falsification approaches:

- **Inner Approximation-Based Falsification** for proving reachability of desired states and methods for nonlinear systems.
- **Simulation-Based Falsification** for handling discrepancies in simulated trajectories, especially in complex systems.
- **Motion Planning-Based Falsification** involves using algorithms like Rapidly Growing Random Trees (RRT) to systematically explore the space of possible system behaviors, seeking scenarios that violate safety or performance criteria
- **Optimization-Based Falsification** leverages mathematical optimization techniques to systematically search for scenarios or parameter combinations that lead to violations of safety or performance properties. This method is often used for deterministic systems

Future directions

- Expanding Model Support
  - This work supports both deterministic and nondeterministic models.
  - Can enhance the capability to analyze any arbitrary nonlinear systems.

- Integration with Verification Framework
  - Exploring integration of path falsification into the Counterexample-Guided Abstraction Refinement (CEGAR) loop.
  - Strengthening the verification and falsification processes.

- Leveraging Reinforcement Learning
  - Investigating the combination of reinforcement learning with our classification model-based DFO in the falsification of Cyber-Physical Systems (CPS).
  - Enhancing the adaptability and performance of the falsification approach.

Conclusion

- In this work, author introduced a path-oriented, derivative-free approach for safety falsification for a large scope of CPS, that could be modeled as hybrid automata for both nonlinear and nondeterministic behavior.
- Two-layered methodology efficiently handled the vast search space of nondeterministic systems and the nonlinear dynamics of CPS.
- Author also devised lightweight techniques for pruning the search space, enhancing efficiency.
- This approach, implemented in the PDF prototype tool, successfully falsified safety properties in both deterministic and nondeterministic benchmarks, achieving high success rates and satisfactory efficiency.

References

Jiawan Wang, Lei Bu, Shaopeng Xing, Xuandong Li “PDF: Path-Oriented, Derivative-Free Approach for Safety Falsification of Nonlinear and Nondeterministic CPS” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems .

E. A. Lee and S. A. Seshia, Introduction to Embedded Systems: A Cyber-Physical Systems Approach, MA, USA:MIT Press, 2017.

R. Alur et al., "The algorithmic analysis of hybrid systems", Theor. Comput. Sci., vol. 138, no. 1, pp. 3-34, 1995.

E. Asarin, T. Dang and O. Maler, "The d/dt tool for verification of hybrid systems", Proc. Int. Conf. Comput.-Aided Verif. (CAV), pp. 365-370, 2002.

Implementation

algo2.ipynb

File Edit View Insert Runtime Tools Help Changes will not be saved

+ Code + Text Copy to Drive

```python
def GENPREFIX(candidatePath, numSimulations):
    index = len(candidatePath) - 1  # Initialize the index to the last location

    for i in range(1, 4):  # Check location sets with size 1-3
        for j in range(len(candidatePath)):
            locationSet = candidatePath[j:j + i]

            sat = False  # Initialize satisfiability flag

            for k in range(numSimulations):  # Check all simulations
                # Assuming you have a function check_constraints() to check if all constraints are TRUE for the location set
                if check_constraints(locationSet):
                    sat = True
                    break  # Exit the loop if constraints are met

            if not sat:  # Update to the smallest one
                index = min(index, locationSet[i - 1])

    # Create and return the Shortest Hardly Feasible Path Prefix
    shortestPrefix = candidatePath[:index + 1]
    return shortestPrefix

def check_constraints(locationSet):
    # Implement your constraint checking logic here
    # For example, check if x >= 5
    for location in locationSet:
        if location < 5:
            return False
    return True

# Example usage
candidatePath = [1, 2, 3, 4, 5, 6, 7]  # Replace with your candidate path
numSimulations = 10  # Replace with the desired number of simulations

shortestPrefix = GENPREFIX(candidatePath, numSimulations)

# Display the result
print("Shortest Hardly Feasible Path Prefix:", shortestPrefix)
```

```python
import random

# Define the function to evaluate the dissatisfactory degree
def F(δ):
    # Dissatisfactory degree function: x + 5
    return δ + 5

# Define the function to classify solutions
def classify_solutions(S, Y):
    # Implement your solution classification logic here
    # For example, return S+ and S- based on S and Y
    S_plus = S  # Positive solutions (for example, lower dissatisfactory degree)
    S_minus = []  # Negative solutions (for example, higher dissatisfactory degree)
    return S_plus, S_minus

# Define the MODELREFINE function
def MODELREFINE(X, δ_plus, S_minus):
    H = list(X)  # Initialize the new model as a copy of the domain
    while any(δ_minus in S_minus for δ_minus in H):
        d = random.randint(0, len(X[0]) - 1)  # Select a dimension
        S_minus_l = [δ_minus for δ_minus in S_minus if δ_minus[d] > δ_plus[d]]
        S_minus_s = [δ_minus for δ_minus in S_minus if δ_minus[d] < δ_plus[d]]
        if len(S_minus_l) >= len(S_minus_s):
            r = random.uniform(δ_plus[d], min(δ[d] for δ in S_minus_l))
            H = [δ for δ in H if δ[d] <= r]
        if len(S_minus_l) <= len(S_minus_s):
            r = random.uniform(max(δ[d] for δ in S_minus_s), δ_plus[d])
            H = [δ for δ in H if δ[d] >= r]
    return H

# Define the SAMPLE function
def SAMPLE(H, δ_plus, n, D):
    δ = list(δ_plus)  # Initialize a new solution as a copy of δ_plus
    for _ in range(n):  # Mutate n times
        d = random.randint(0, D - 1)  # Select a dimension
        δ[d] = random.uniform(δ_plus[d] - 1, δ_plus[d] + 1)  # Adjust the range as needed
    return δ  # Return the new solution

# Define the main optimization function
def OPTIMIZE(X, F, N, m, n):
    S, Y = [], []  # Solution set, evaluation set
```

- **Implementation of classification model based derivative free optimization algorithm in python**
- **Implementation of algorithm to identify shortest hardly feasible path in python**

Critical Analysis

- We looked in to authors' claim that due to absence of publicly available tool for falsification of non-linear and non-deterministic hybrid automata, they could do comparative analysis for only deterministic scenarios with S-TaLiRo and BREACH. However, we observed that these tools though have their limitations, such as S-TaLiRo focuses on software testing, and BREACH need states to be defined as boolean expression, do not limit non-deterministic modelling. Result however are not very reliable for such scenarios.
- Another tool SpaceEx could have been used for comparison. It facilitates use of falsification for non-deterministic hybrid automata.
- Authors could not be reached to have their comments on our observations.

Contribution of Members

Ser	Name	Roll No	Contribution (%)
1	Himanshu Karnatak	231110017	14.3
2	LtCdr Sunil Kumar	231110025	14.3
3	Priyanshu Chaurasiya	210780	14.3
4	Pradeep Kumar Bagri	210734	14.3
5	Ashish Saini	210212	14.3
6	Harsh Khandelwal	210407	14.3
7	Palak Parashar	210691	14.3

Thank You

Implementation & Verification

```
# Define the main optimization function
def OPTIMIZE(X, F, N, m, n):
    S, Y = [], []  # Solution set, evaluation set
    S_plus, S_minus = list(X), []  # Positive and negative solutions
    δ_optimal = []  # Optimal solution
    F_optimal = float('inf')  # Optimal evaluation

    for _ in range(N):
        for _ in range(m):
            δ_plus = random.choice(S_plus)
            H = MODELREFINE(X, δ_plus, S_minus)
            δ_samples = [SAMPLE(H, δ_plus, n, 1) for _ in range(len(X))]
            Y = [F(δ_i) for δ_i in δ_samples]
            S, S_minus = classify_solutions(S, Y)
            δ_optimal = min(S + [δ_optimal], key=lambda δ: F(δ))
            F_optimal = F(δ_optimal)

    return δ_optimal, F_optimal

# Example usage
X = [0, 1, 2, 3]
N = 10
m = 5
n = 3

optimal_solution, optimal_evaluation = OPTIMIZE(X, F, N, m, n)
print("Optimal Solution:", optimal_solution)
print("Optimal Evaluation:", optimal_evaluation)
```